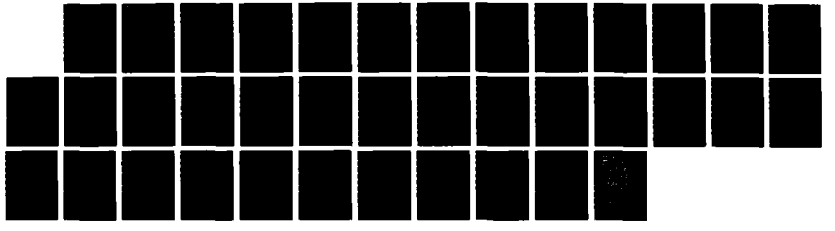
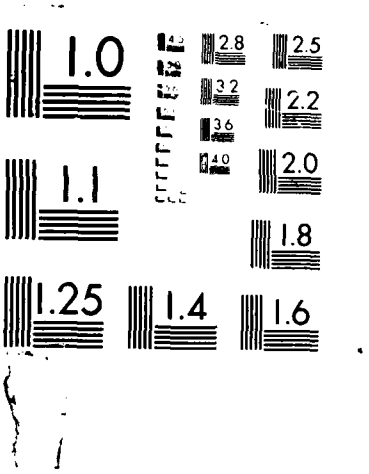
AD-A189 269 ADA (TRADE NAME) VALIDATION SUMMARY REPORT: AT&T UNIX 1/1
ADA RELEASE 1.0 3B2/400 WITH MAU(V) INFORMATION SYSTEMS
AND TECHNOLOGY CENTER W-P AFB OH ADA VALI.. 21 FEB 87
UNCLASSIFIED F/G 12/5 NL




1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

AVF Control Number: AVF-VSR-57.0587
86-11-25-ATT

Ada® COMPILER VALIDATION SUMMARY REPORT:

AT&T
UNIX Ada, Release 1.0
3B2/400 with MAU

Completion of On-Site Testing:
21 February 1987

Prepared By:
Ada Validation Facility
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETEING FORM
1. REPORT NUMBER | 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: AT&T, UNIX Ada, Release 1.0, 3B2/400	5. TYPE OF REPORT & PERIOD COVERED 21 Feb.'87 to 21 Feb '88 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Wright-Patterson AFB	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Wright-Patterson AFB, OH 45433066503	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081ASD/SIOL	12. REPORT DATE 21 Feb.'87 13. NUMBER OF PAGES 33
14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office) Wright-Patterson AFB	15. SECURITY CLASS (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20. If different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

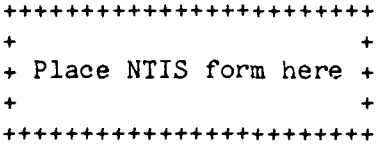
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See Attached.

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
1 JAN 73 S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)


Place NTIS form here

Ada® Compiler Validation Summary Report:

Compiler Name: UNIX Ada, Release 1.0

Host:	Target:
3B2/400 with MAU under UNIX System V, Release 3.1	3B2/400 with MAU under UNIX System V, Release 3.1

Testing Completed 21 February 1987 Using ACVC 1.8

This report has been reviewed and is approved.

Georgeanne C Chitwood

Ada Validation Facility
Georgeanne Chitwood
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

John F Kramer

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA

Virginia L. Castor

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the UNIX Ada, Release 1.0, using Version 1.8 of the Ada® Compiler Validation Capability (ACVC). The UNIX Ada is hosted on a 3B2/400 with MAU operating under UNIX System V, Release 3.1. Programs processed by this compiler may be executed on a 3B2/400 with MAU operating under UNIX SYSTEM V, Release 3.1.

On-site testing was performed 16 February 1987 through 21 February 1987 at AT&T, Summit NJ, under the direction of the Ada Validation Facility (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2210 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as the 170 executable tests that make use of floating-point precision exceeding that supported by the implementation, were not processed. After the 2210 tests were processed, results for Class A, C, D, or E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 56 of the processed tests determined to be inapplicable. The remaining 2154 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	89	252	334	243	161	97	136	261	109	32	217	223	2154
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	27	73	86	4	0	0	3	1	21	0	1	10	226
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

® Ada is a registered trademark of the United States Government (Ada Joint Program Office).

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from characteristics of particular operating systems, hardware, or implementation strategies. All of the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any unsupported language constructs required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc., under the direction of the AVF according to policies and procedures established by the Ada Validation Organization (AVO). On-site testing was conducted from 16 February 1987 through 21 February 1987 at AT&T, Summit NJ.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Ada Validation Facility
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, FEB 1983.

2. Ada Validation Organization: Procedures and Guidelines, Ada Joint Program Office, 1 JAN 1987.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., DEC 1984.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. A set of programs that evaluates the conformity of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.
Ada Standard	ANSI/MIL-STD-1815A, February 1983.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.
AVO	The Ada Validation Organization. In the context of this report, the AVO is responsible for setting procedures for compiler validations.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	A test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.

Inapplicable test	A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	A test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	A test found to be incorrect and not used to check conformity to the Ada language specification. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. However, no checks are performed during execution to see if the test objective has been met. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers

permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of these units is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: UNIX Ada, Release 1.0

ACVC Version: 1.8

Certificate Expiration Date: 31 March 1988

Host Computer:

Machine:	3B2/400 with MAU
Operating System:	UNIX System V Release 3.1
Memory Size:	4 megabytes

Target Computer:

Machine:	3B2/400 with MAU
Operating System:	UNIX System V Release 3.1
Memory Size:	4 megabytes

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

- Capacities.

  The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types SHORT_INTEGER, LONG_FLOAT, and BYTE_INTEGER in the package STANDARD. (See tests B86001C and B86001D.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

- Array types.

  An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT.

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array objects are declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array objects are declared. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when array objects are assigned. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

In assigning record types with discriminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Aggregates.

In the evaluation of a multi-dimensional aggregate, index subtype checks appear to be made as choices are evaluated. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are not evaluated before being checked for identical bounds. (See test E43212B.)

All choices are not evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

. Functions.

An implementation may allow the declaration of a parameterless function and an enumeration literal having the same profile in the same immediate scope, or it may reject the function declaration. If it accepts the function declaration, the use of the enumeration literal's identifier denotes the function. This implementation accepts the declaration. This behavior is contrary to AI-00330, which was recently approved by the Director, AJPO. This test will be reclassified as a Class B test in future versions of the ACVC. (See test E66001D.)

. Representation clauses.

The Ada Standard does not require an implementation to support representation clauses. If a representation clause is not supported, then the implementation must reject it. While the operation of representation clauses is not checked by Version 1.8 of the ACVC, they are used in testing other language features. This implementation rejects 'SIZE, 'STORAGE_SIZE for collections, and 'SMALL clauses. Enumeration representation clauses, including those that specify noncontiguous values, appear not to be supported. (See tests C55B16A, C87B62A, C87B62B, C87B62C, and BC1002A.)

. Pragmas.

The pragma INLINE is not supported for procedures. The pragma INLINE is not supported for functions. (See tests CA3004E and CA3004F.)

. Input/output.

The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants. The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, AE2101H, CE2201D, CE2201E, and CE2401D.)

An existing text file can be opened in OUT_FILE mode, can be created in OUT_FILE mode, and can be created in IN_FILE mode. (See test EE3102C.)

More than one internal file can be associated with each external file for text I/O for both reading and writing. (See tests CE3111A..E (5 tests).)

More than one internal file can be associated with each external file for sequential I/O for both reading and writing. (See tests CE2107A..F (6 tests).)

More than one internal file can be associated with each external file for direct I/O for both reading and writing. (See tests CE2107A..F (6 tests).)

An external file associated with more than one internal file can be deleted. (See test CE2110B.)

Temporary sequential files are not given a name. Temporary direct files are not given a name. (See tests CE2108A and CE2108C.)

. Generics.

Generic declarations and bodies cannot be compiled in separate compilations. (See tests BA1011C, CA1012A, CA2009C, CA2009F, LA5008A..H (8 tests), LA5008J, LA5008M..N (2 tests), and BC3205D.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.8 of the ACVC contains 2399 tests. When validation testing of UNIX Ada was performed, 19 tests had been withdrawn. The remaining 2380 tests were potentially applicable to this validation. The AVF determined that 226 tests were inapplicable to this implementation, and that the 2154 applicable tests were passed by the implementation.

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	69	860	1164	17	11	33	2154
Failed	0	0	0	0	0	0	0
Inapplicable	0	7	204	0	2	13	226
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	89	252	334	243	161	97	136	261	109	32	217	223	2154
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	27	73	86	4	0	0	3	1	21	0	1	10	226
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

3.4 WITHDRAWN TESTS

The following 19 tests were withdrawn from ACVC Version 1.8 at the time of this validation:

C32114A	C41404A	B74101B
B33203C	B45116A	C87B50A
C34018A	C48008A	C92005A
C35904A	B49006A	C940ACA
B37401A	B4A010C	CA3005A..D (4 tests)
		BC3204C

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. For this validation attempt, 226 tests were inapplicable for the reasons indicated:

- B23003D..F (3 tests), C23003A..C (3 tests), C23003G..J (4 tests), and C24003A..C (3 tests) test the maximum line length of the compiler. This compiler has no maximum line length.
- C34001E, B52004D, B55B09C, and C55B07A use LONG_INTEGER which is not supported by this compiler.
- C34001F and C35702A use SHORT_FLOAT which is not supported by this compiler.

- C55B16A makes use of an enumeration representation clause containing noncontiguous values which is not supported by this compiler.

- C87B62A..C (3 tests) use length clauses which are not supported by this compiler. C87B62B uses the length clause 'STORAGE_SIZE for tasks and for access types. The length clause 'STORAGE_SIZE for tasks is accepted and the length clause 'STORAGE_SIZE for access types is rejected during compilation. The length clauses in tests C87B62A and C87B62C are rejected during compilation.

- C96005B checks implementations for which the smallest and largest values in type DURATION are different from the smallest and largest values in DURATION's base type. This is not the case for this implementation.

- Tests BA1011C, CA1012A, CA2009C, CA2009F, LA5008A..H (8 tests), LA5008J, LA5008M..N (2 tests), and BC3205D compiles generic declarations and bodies in separate compilation units. Separate compilation of generic specifications and bodies is not supported by this compiler.

- CA3004E, FA3004C, and LA3004A use INLINE pragma for procedures which is not supported by this compiler.

- CA3004F, FA3004D, and LA3004B use INLINE pragma for functions which is not supported by this compiler.

- CE2201A and CE2401B read access types from a file. This implementation raises DATA_ERROR when an attempt is made to read access types from a file; the AVO ruled that this behavior is acceptable. The output from these tests was examined, and it showed that I/O for all other types was performed correctly.

- CE2107C..D (2 tests), CE2108A..D (4 tests), and CE3112A..B (2 tests) are inapplicable because these tests expect a temporary file to have a name. This implementation raises the exception USE_ERROR when the NAME function is called with a temporary file as its argument.

- The following 170 tests require a floating-point accuracy that exceeds the maximum of 15 supported by the implementation:

C24113L..Y (14 tests)
C35705L..Y (14 tests)
C35706L..Y (14 tests)
C35707L..Y (14 tests)
C35708L..Y (14 tests)
C35802L..Y (14 tests)
C45241L..Y (14 tests)
C45321L..Y (14 tests)
C45421L..Y (14 tests)

C45424L..Y (14 tests)
C45521L..Z (15 tests)
C45621L..Z (15 tests)

3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of smaller tests that contain the undetected errors. These splits are then compiled and examined. The splitting process continues until all errors are detected by the compiler or until there is exactly one error per split. Any Class A, Class C, or Class E test that cannot be compiled and executed because of its size is split into a set of smaller subtests that can be processed.

Splits were required for 59 Class B tests.

B26005A	B32103A	B32201A
B33101A	B33102B	B33102C
B33102D	B33102E	B33201A
B33201C	B33202A	B33202C
B33203A	B33301A	B36201A
B37101A	B37201A	B37302A
B38008A	B38103A	B38103B
B38103C	B41201A	B41202A
B43201A	B44001A	B44002A
B48002A	B48002D	B54A01L
B54A20A	B54A25A	B55A01A
B56001A	B64001A	B64003A
B67001A	B67001B	B67001C
B67001D	B67004A	B74001A
B74001B	B74104A	B74301A
B83A06B	B91003B	B95001A
B95004B	B95007B	B95032A
B95081A	B950BAA	B97103A
BA1101C	BA2001A	BB2002A
BC1009A	BC3204D	

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the UNIX Ada was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the UNIX Ada using ACVC Version 1.8 was conducted on-site by a validation team from the AVF. The configuration consisted of a 3B2/400 with MAU operating under UNIX System V, Release 3.1.

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring splits during the prevalidation testing were included in their split form on the magnetic tape.

The contents of the magnetic tape were not loaded directly onto the host computer. The tapes were loaded onto an AT&T 3B20 computer and the files were renamed to conform to UNIX Ada requirements. The 3B2/400 with MAU and the 3B20 are connected by 3BNet (a form of ethernet) which was used to transfer the files to the 3B2/400. After the test files were loaded to disk, the full set of tests was compiled and linked on the 3B2/400 with MAU, and all executable tests were linked and run. The results were sent back to the original 3B20 via 3BNet and printed.

The compiler was tested using command scripts provided by AT&T and reviewed by the validation team.

Tests were compiled, linked, and executed (as appropriate) using a single host computer and target computer. Test output and compilation listings were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

The validation team arrived at AT&T, Summit NJ on 16 February 1987, and departed after testing was completed on 21 February 1987.

APPENDIX A

COMPLIANCE STATEMENT

AT&T has submitted the following compliance statement concerning the UNIX Ada.

DECLARATION OF CONFORMANCE

Compiler Implementor: AT&T
Ada®Validation Facility: ASD/SCOL, Wright-Patterson AFB, OH
Ada Compiler Validation Capability (ACVC) Version: 1.8

Base Configuration

Base Compiler Name:	UNIX Ada	Version:	Release 1.0
Host Architecture ISA:	3B2/400 with MAU	OS&VER #:	UNIX System V Release 3.1
Target Architecture ISA:	3B2/400 with MAU	OS&VER #:	UNIX System V Release 3.1

Implementor's Declaration

I, the undersigned, representing AT&T, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that AT&T is the owner of record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.

Philip E. Brown Date: 2-19-87
AT&T
Philip E. Brown, Member of Technical Staff

Owner's Declaration

I, the undersigned, representing AT&T, take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A. I have reviewed the Validation Summary Report for the compiler(s) and concur with the contents.

Philip E. Brown Date: 2-19-87
AT&T
Philip E. Brown, Member of Technical Staff

---

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation classes. The implementation-dependent characteristics of the UNIX Ada, Release 1.0, are described in the following sections which discuss topics in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Implementation-specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is

     ...

     type INTEGER is range -2_147_483_648 .. 2_147_483_647;
     type SHORT_INTEGER is range -32768 .. 32767;
     type BYTE_INTEGER is range -128 .. 127;

     type FLOAT is digits 6 range -7.92281E28 .. 7.92281E28;
     type LONG_FLOAT is digits 15
               range -6.58201822928482E63 .. 6.58201822928482E63;

     type DURATION is delta 1.0/128 range -86400.0 .. 86400.0;

     ...

end STANDARD;
```

APPENDIX F OF THE Ada STANDARD

B.1 UNIX ADA PRAGMAS

The UNIX Ada compiler recognizes all language-defined pragmas and one more as well. Appendix C of this manual summarizes all the pragmas, noting those that have an effect on the UNIX Ada system and those that do not. All the pragmas having an effect on the UNIX Ada system are thoroughly explained in Chapther 13 on Ada pragmas and in the pertinent chapters throughout the manual. This section briefly describes the pragma implemented as an enhancement to the UNIX Ada system, namely pragma LIBRARY.

B.1.1 Pragma LIBRARY

This special UNIX Ada pragma is one way to specify in what program library a required compilation unit is to be found. Its format is

```
pragma LIBRARY(UNIX_Ada_Library_Alias, Ada_Compilation_Unit_Name);
```

For details on its placement in your program and examples of its use, consult Chapter 13 on Ada pragmas or Appendix C, the summary of Ada pragmas. For alternative methods of locating a required compilation unit, turn to Chapter 4 on the -I and -U options of the ada command.

B.2 IMPLEMENTATION-DEPENDENT ATTRIBUTES

Because the existence of implementation-dependent attributes generally impedes the portability of Ada programs, UNIX Ada seeks to limit their number. At present, there are none.

B.3 THE PACKAGE SYSTEM

The package SYSTEM is partly specified in LRM 13.7. Its complete implementation-dependent specification within UNIX Ada is as follows:

```
package SYSTEM is
     type address is new integer;
     type name is (ATT_3B2_400);

     SYSTEM.NAME: constant NAME := ATT_3B2_400;
     STORAGE_UNIT: constant := 8;
     MEMORY_SIZE: constant := 16_777_216;  -- 16 megabytes

     -- System dependent named numbers

     MIN_INT: constant := -2_147_483_648;
     MAX_INT: constant := 2_147_483_647;
     MAX_DIGITS: constant := 15;
     MAX_MANTISSA: constant := 31;
     FINE_DELTA: constant := 1.0 / 1_073_741_824;
     TICK: constant := 1.0 / 100;

     -- Other system dependent declarations

     subtype PRIORITY is INTEGER range 1 .. 14;
end SYSTEM;
```

The type ADDRESS is new INTEGER. The parent type INTEGER ensures that arithmetic may be done with addresses; making ADDRESS a derived type (with the indicator new) ensures that addresses are not inadverently mixed with other INTEGER types. See Chapter 8 for more information.

The type NAME is an enumeration type consisting of the single value ATT_3B2_400.

The constant SYSTEM_NAME equals this sole value ATT_3B2_400 of the type NAME.

The constant STORAGE_UNIT is 8; this is the number of bits in one byte, the basic storage unit of the system. The type of STORAGE_UNIT is universal integer.

The constant MEMORY_SIZE is 16 * 1024 * 1024 bytes, or equivalently 16 megabytes (16,777,216 bytes). Its type is also universal integer. The constant indicates the maximum size for a UNIX system process, and hence the maximum size for an executing Ada program. Main memory on a particular configuration will be less, but with paging from disk storage, the UNIX system provides up to four gigabytes (4 * 1024 * 1024 * 1024 bytes) of virtually addressable storage space.

The constant MIN_INT equals -(2**31), or equivalently -2,147,483,648 the smallest or most negative integer value representable on the system.

The constant MAX_INT equals 2**31 - 1, or equivalently 2,147,483,647 the largest integer value representable on the system.

The constant MAX_DIGITS is 15. It is the largest number of significant decimal digits expressible in a floating point number.

The constant MAX_MANTISSA is 31. It is the largest number of binary digits that a fixed point number can use in its representation.

The constant FINE_DELTA equals 2**(-30) or equivalently 1.0 / 1,073,741,824; it is the smallest delta value for any fixed point type on the system in the range -1.0 .. 1.0. For more on fixed point numbers, see Chapter 9.

The constant TICK, equal to 1.0 / 100th of a second, is the clock period, the smallest unit of time measurable by the system clock. It differs from both the value of DURATION'SMALL and the smallest parameter value associated with Alarm, the smallest delay possible on the system. For futher discussion, see Chapter 12 on tasking.

The subtype PRIORITY encompasses the integer values from 1 through 14. Pragma PRIORITY sets the priority of a task with values drawn from this subtype. For more on the pragma, see Chapter 12 on tasking and Chapter 14 on pragmas.

B.4 REPRESENTATION CLAUSES IN UNIX ADA

A representation clause in Ada is a specification indicating features of the underlying hardware to be used for representing an Ada type, object, or other entity. UNIX Ada faithfully acts upon the following representation clauses:

- Ada length clauses with T'STORAGE_SIZE, where T is a task type, which specify the number of storage units (bytes) to be allocated for the activation, not the code, of task objects of the task type (LRM 13.2).

In all other cases, UNIX Ada prints a warning message.

B.5 IMPLEMENTATION-GENERATED NAMES DENOTING IMPLEMENTATION-DEPENDENT RECORD COMPONENTS

To promote Ada program portability, UNIX Ada does not provide access to implementation-generated names for implementation-dependent record components. In general, UNIX Ada does not follow the dictates of a record representation clause; instead, it issues a warning message. See LRM 13.4.

B.6 INTERPRETATION OF EXPRESSIONS IN ADDRESS CLAUSES

As stated a moment ago, UNIX Ada does not support address clauses, including those for interrupts (LRM 13.5). When UNIX Ada encounters an address clause, it prints an error message saying that it does not recognize it. Instead, UNIX Ada places the entity in question at the most suitable location possible.

B.7 RESTRICTIONS ON UNCHECKED CONVERSIONS

UNIX Ada fully implements the Ada generic function UNCHECKED_CONVERSION (LRM 13.10, especially 13.10.2). Because there are no restrictions on its source and target types, care must be exercised in its use.

B.8 IMPLEMENTATION-DEPENDENT CHARACTERISTICS OF UNIX ADA INPUT-OUTPUT PACKAGES

With every file type, if the main program terminates before all of its files have been closed, UNIX Ada will automatically close them. Whatever temporary files remain will be deleted (LRM 14.1/7).

UNIX Ada itself does not arbitrarily limit the number of file objects you may associate with a given external file through the procedures OPEN and CREATE. The UNIX system, however, does impose a limit of 20 (LRM 14.1/13).

The type FILE_TYPE is a private type, defined in the private sections of the I/O packages as the type access integer.

The fourth and last parameter of the CREATE and OPEN procedures is the string parameter FORM. In UNIX Ada, you may use the parameter only with CREATE to specify the owner of the file to be created, the group to which the owner belongs, and the read and write permissions to be associated with the file. See Chapter 10 for more information on this parameter.

With SEQUENTIAL_IO and DIRECT_IO, if you output an access type, UNIX Ada treats it as if it were the output of an integer. On the other hand, input of an access type raises the exception DATA_ERROR.

The READ procedure does not check if the element read can be interpreted as a value of the instantiated type ELEMENT_TYPE (LRM 14.2.2).

The subtype FIELD in TEXT_IO states the range of values for the width of fields on input and output; in UNIX Ada, its range is 0 .. 36.

In a text file, the line-feed character STANDARD.ASCII.LF is the line terminator, and the form-feed character STANDARD.ASCII.FF is the page terminator. UNIX Ada uses the end-of-file condition (determined by keeping track of the number of bytes in the file) as the sole file terminator; it does not automatically store the line and page terminators at the end of a

file.

There is no LOW_LEVEL_IO package.

UNIX Ada provides several enhancements to the standard Ada I/O packages. The packages UA_SEQUENTIAL_IO, UA_DIRECT_IO, and UA_TEXT_IO contain subprograms that translate between UNIX system file descriptors and Ada FILE_TYPE objects. The package UA_TEXT_IO also contains subprograms for returning and setting the UNIX standard error file. See Chapter 11 for more comprehensive I/O information.

B.9 ADDITIONAL UNIX ADA IMPLEMENTATION-DEPENDENT FEATURES

B.9.1 Compiling Generic Units

The UNIX Ada system requires that a generic declaration be submitted for compilation in the same file as its corresponding body and subunits, if any (LRM 10.3).

B.9.2 MACHINE_CODE Package

UNIX Ada does not support this package; all CODE statements are illegal.

B.9.3 Unlimited Parameter Sizes

UNIX Ada puts virtually no limits on the size of the various parameters. The sole exception pertains to the length of identifiers.

- The length of the internal name which UNIX Ada uses is limited to 512 characters. This internal name is formed by stringing together the program names of all the program units enclosing the declaration of the object in question and separating them with underscores. The maximal length for a particular identifier therefore depends in part on the context in which it is declared.

Whatever other limits arise are a function of the storage available during runtime and the current application program. In particular, there are NO FORMAL LIMITS (those stemming from UNIX Ada) on the number of:

- lines in an Ada source file;
- parameters in a procedure, task entry, or function call;
- discriminants for a record type;

- dimensions in an array type;
- Ada frames that an exception can propagate;
- fields in a record type;
- library units and/or subunits in an executable Ada program;
- enumeration values in an enumeration type.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	(1..199 =>'A', 200 =>'1')
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..199 =>'A', 200 =>'2')
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..99 =>'A', 100 =>'3', 101..200 =>'A')
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..99 =>'A', 100 =>'4', 101..200 =>'A')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..197 =>'0', 198..200 =>"298")

Name and Meaning	Value
$BIG_REAL_LIT A real literal that can be either of floating- or fixed-point type, has value 690.0, and has enough leading zeroes to be the size of the maximum line length.	(1..194 =>'A', 195..200 =>"690.0E1")
$BLANKS A sequence of blanks twenty characters fewer than the size of the maximum line length.	(1..180 =>' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2_147_483_647
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz!$%?@[\]^`{}~"
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	36
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters, or is too long if no invalid characters exist.	xx/yy
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character, or is too long if no wild card character exists.	file0123456789012345
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST if any, otherwise any value in the range of DURATION.	100_000.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST, if such a value exists.	10_000_000.0

Name and Meaning	Value
$ILLEGAL_EXTERNAL_FILE_NAME1 An illegal external file name.	xx/yy
$ILLEGAL_EXTERNAL_FILE_NAME2 An illegal external file name that is different from $ILLEGAL_EXTERNAL_FILE_NAME1.	MUCH-TOO-LONG-NAME-FOR-A-FILE
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-2_147_483_648
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	2_147_483_647
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST if any, otherwise any value in the range of DURATION.	-100_000.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST, if such a value exists.	-10_000_000.0
$MAX_DIGITS The universal integer literal whose value is the maximum digits supported for floating-point types.	15
$MAX_IN_LEN The universal integer literal whose value is the maximum input line length permitted by the implementation.	200
$MAX_INT The universal integer literal whose value is SYSTEM.MAX_INT.	2_147_483_647

Name and Meaning	Value
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER if one exists, otherwise any undefined name.	BYTE_INTEGER
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	8#777777777776#
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non-ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 19 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

- C32114A: An unterminated string literal occurs at line 62.

- B33203C: The reserved word "IS" is misspelled at line 45.

- C34018A: The call of function G at line 114 is ambiguous in the presence of implicit conversions.

- C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR instead of CONSTRAINT_ERROR as expected in the test.

- B37401A: The object declarations at lines 126 through 135 follow subprogram bodies declared in the same declarative part.

- C41404A: The values of 'LAST and 'LENGTH are incorrect in the if statements from line 74 to the end of the test.

- B45116A: ARRPRIBL1 and ARRPRIBL2 are initialized with a value of the wrong type--PRIBOOL_TYPE instead of ARRPRIBOOL_TYPE--at line 41.

- C48008A: The assumption that evaluation of default initial values occurs when an exception is raised by an allocator is incorrect according to AI-00397.

WITHDRAWN TESTS

- B49000A: Object declarations at lines 41 and 50 are terminated incorrectly with colons, and end case; is missing from line 42.

- B4A010C: The object declaration in line 18 follows a subprogram body of the same declarative part.

- B74101B: The begin at line 9 causes a declarative part to be treated as a sequence of statements.

- C87B50A: The call of "/=" at line 31 requires a use clause for package A.

- C92005A: The "/=" for type PACK.BIG_INT at line 40 is not visible without a use clause for the package PACK.

- C940ACA: The assumption that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program, is erroneous.

- CA3005A..D (4 tests): No valid elaboration order exists for these tests.

- BC3204C: The body of BC3204C0 is missing.

END

DATE

FILMED

MARCH

1988

DTIC